FIAP ON

DESENVOLVIMENTO DE APPS
– PARTE 1 (ANDROID)

TRANSIÇÕES, CICLO DE VIDA E ESTADOS

RAFAEL ACIOLY DE OLIVEIRA

3

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE CÓDIGOS-FONTE

SUMÁRIO

3 TRANSIÇÕES, CICLO DE VIDA E ESTADOS

Nossas aplicações têm diversos comportamentos desde o momento em que abrimos o app até o encerramento dele, e é isto que você aprenderá neste capítulo: o ciclo de vida da activity!

Activity é uma classe que representa uma tela/interface do aplicativo e é responsável por tratar os eventos gerados nela. Podem ser organizadas em blocos totalmente reutilizáveis para serem compartilhadas entre diferentes aplicações.

Dentro de um projeto, podemos ter diversas Activities e, para que funcione corretamente, cada uma delas deve ser declarada no arquivo *AndroidManifest.xml (*exemplificado de forma resumida no Código-fonte *"*Declaração de Activity no manifesto"), responsável por administrar e informar ao sistema operacional que elas existem e como podem ser utilizadas.

```
<manifest ... >
<application ... >
<activity android:name=".MainActivity" />
...
</application ... >
...
</manifest >
```

Código-fonte 3.1 – Declaração de Activity no manifesto (AndroidManifest.xml)
Fonte: Elaborado pelo autor (2019)

3.1 Como uma Activity funciona

Uma Activity possui estados de funcionamento: são criadas, iniciadas, pausadas, reiniciadas e destruídas. Os eventos que se passam em uma Activity são conhecidos como *ciclo de vida* e incluem esses estados que, para cada um deles, existe um método de retorno dentro da Activity, sendo eles: *onCreate, onStart, onRestart, onResume, onPause, onStop e onDestroy*. A Figura "Ciclo de vida de uma Activity" ilustra os estados de uma Activity assim como o respectivo fluxo.

Para cada estado de uma Activity, o desenvolvedor pode tomar ações referente às necessidades de funcionamento do aplicativo.

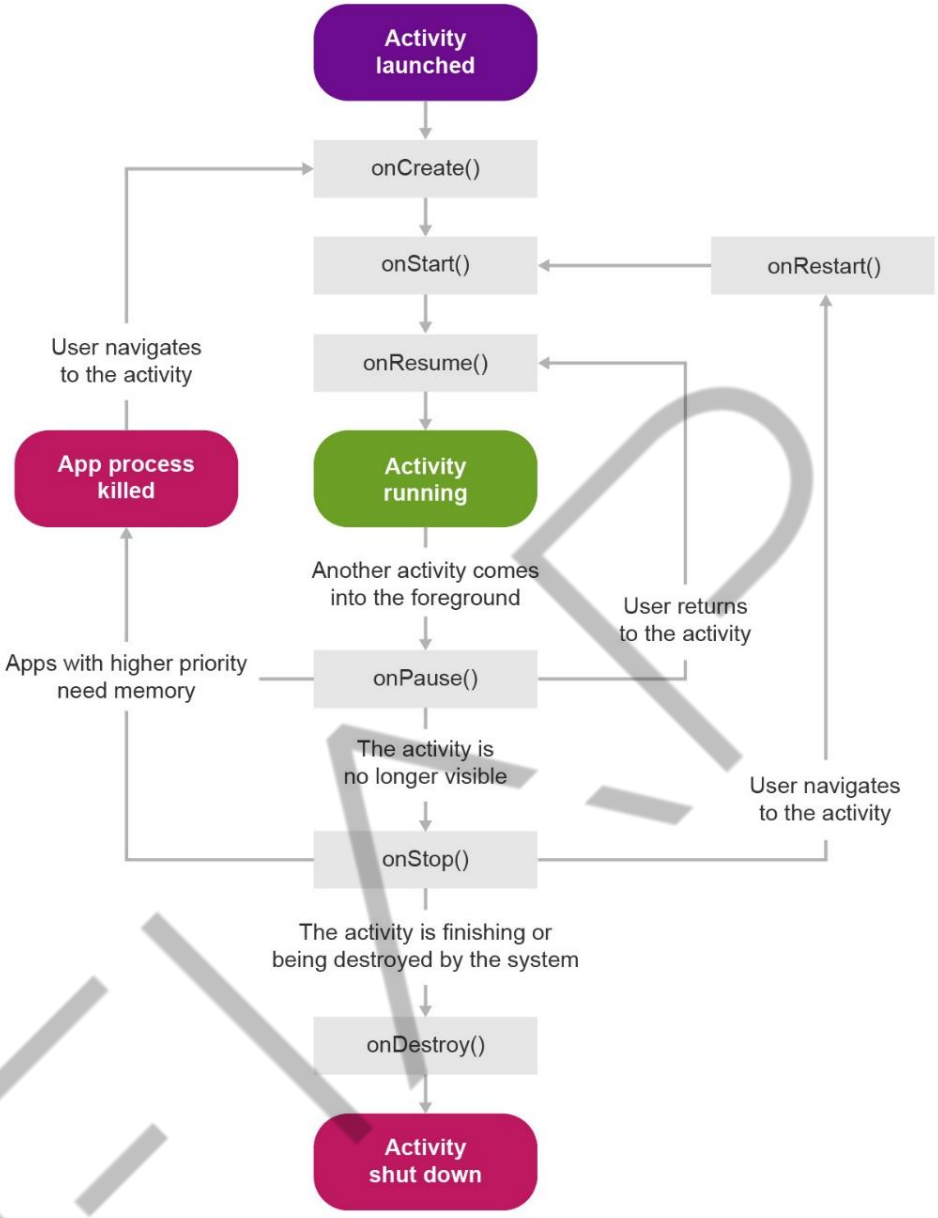

Figura 3.1 – Ciclo de vida de uma Activity
Fonte: Guia de atividades Android (2019)

É muito importante que um desenvolvedor conheça o ciclo, dada a oportunidade de colocar uma determinada ação de acordo com o estado. O Quadro “Descrição dos métodos do ciclo de vida” descreve cada estado do ciclo de vida, quais ações podem ser vinculadas por meio de métodos que podem ser escritos na classe que representa a Activity.

Método	Descrição
onCreate()	Chamado quando a atividade é criada pela primeira vez. É onde se deve fazer toda a configuração estática normal — criar exibições, vincular dados a listas etc. Esse método recebe um objeto Bundle (pacote) contendo o estado anterior da atividade, caso esse estado tenha sido capturado (consulte Gravação do estado da atividade). Sempre seguido de onStart().
onRestart()	Chamado depois que a atividade tiver sido interrompida, logo antes de ser reiniciada. Sempre seguido de onStart().
onStart()	Chamado logo antes de a atividade se tornar visível ao usuário. Seguido de onResume() se a atividade for para segundo plano ou onStop() se ficar oculta.
onResume()	Chamado logo antes de a atividade iniciar a interação com o usuário. Nesse ponto, a atividade estará no topo da pilha de atividades com a entrada do usuário direcionada a ela. Sempre seguido de onPause().
onPause()	Chamado quando o sistema está prestes a retomar outra atividade. Esse método normalmente é usado para confirmar alterações não salvas a dados persistentes, animações interrompidas e outras coisas que talvez estejam consumindo CPU e assim por diante. Ele sempre deve fazer tudo bem rapidamente porque a próxima atividade não será retomada até ela retornar. Seguido de onResume() se a atividade retornar para a frente ou de onStop() se ficar invisível ao usuário.
onStop()	Chamado quando a atividade não está mais visível ao usuário. Isso pode acontecer porque ela está sendo destruída ou porque outra atividade (uma existente ou uma nova) foi retomada e está cobrindo-a. Seguido de onRestart() se a atividade estiver voltando a interagir com o usuário ou onDestroy() se estiver saindo
onDestroy()	Chamado antes de a atividade ser destruída. É a última chamada que a atividade receberá. Pode ser chamado porque a atividade está finalizando (alguém chamou finish() nela) ou porque o sistema está destruindo temporariamente essa instância da atividade para poupar espaço. É possível distinguir entre essas duas situações com o método isFinishing()

Quadro 3.1– Descrição dos métodos do ciclo de vida
Fonte: Developer (2019)

O Código-fonte “Arquivo MainActivity.kt” apresenta um exemplo das maneiras que as ações referentes aos estados de uma Activity podem ser invocadas. Este código, quando executado, pode ter o acompanhamento dos momentos nos quais são transitados os estados de uma Activity observando o Logcat (local no qual o Android pode escrever logs em tempo de execução). O método Log.i permite a inserção desses logs.

Observe durante a execução de qualquer aplicativo (podendo até ser um aplicativo com a tela em branco) como os estados da Activity transitam. Para isso, crie um projeto no Android Studio com o código-fonte em Kotlin com uma Empty Activity.

No arquivo MainActivity.kt (caso ele esteja com outro nome, use o arquivo .kt associado), programe as recomendações do Código-fonte “Arquivo MainActivity.kt”.

Repare que, para termos acesso às funcionalidades de Log, precisamos importar a API *Log* com o comando import android.util.Log no topo do nosso código, como mostrado no Código-fonte “Arquivo MainActivity.kt”. Depois disso, use o Logcat, na barra de ferramentas inferior do lado esquerdo, e filtre pelo log desejado como mostra a Figura “Uso do Logcat”.

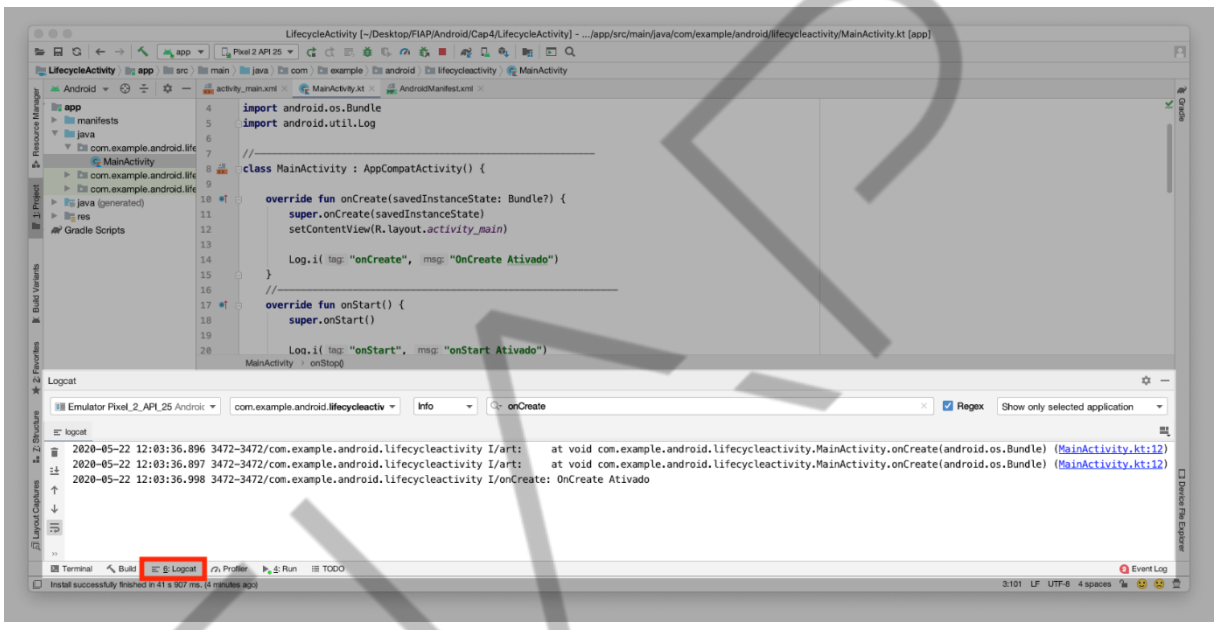

Figura 3.2 – Uso do Logcat
Fonte: Android Studio (2020)

```
import android.util.Log

//-----------------------------------------------------------
class MainActivity : AppCompatActivity() {

        override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        Log.i("onCreate", "OnCreate Ativado")
    }
//-----------------------------------------------------------
        override fun onStart() {
        super.onStart()

        Log.i("onStart", "onStart Ativado")
    }
```

```
//-------------------------------------------------------------
    override fun onResume() {
        super.onResume()

        Log.i("onResume", "onResume Ativado")
    }
//-------------------------------------------------------------
    override fun onPause() {
        super.onPause()

        Log.i("onPause", "onPause Ativado")
    }
//-------------------------------------------------------------
    override fun onStop() {
        super.onStop()

        Log.i("onStop", "onStop Ativado")
    }
//-------------------------------------------------------------
    override fun onDestroy() {
        super.onDestroy()

        Log.i("onDestroy", "onDestroy Ativado")
    }
//-------------------------------------------------------------
    override fun onRestart() {
        super.onRestart()

        Log.i("onRestart", "onRestart Ativado")
    }
}
```

Código-fonte 3.2 – Funcionamento ciclo de vida
Fonte: Elaborado pelo autor (2019)

3.2 Entendendo uma View

View é a classe pai de todos os componentes visuais no Android e suas subclasses são utilizadas para criar a interface gráfica do aplicativo. Simplificando, a View é um retângulo na tela que mostra algum conteúdo. Ela pode ser uma imagem, um pedaço de texto, um botão ou qualquer outra *widget* que o aplicativo possa exibir. Todas as Views juntas formam o layout (que pode ser definido em um arquivo XML ou criado em código-fonte na Activity) o qual define todo o visual do aplicativo. O uso do arquivo XML é mais comum e não descarta a modificação dos elementos de visualização em tempo de execução.

Existem diferentes tipos de **Views** no Android: por exemplo a TextView (view que mostra algum texto), EditText (view que coleta alguma informação digitada pelo usuário), Button (view que mostra um botão) entre muitas outras views. A seguir, vamos aprender como trabalhar com algumas views.

3.3 Componentes Button, TextView e EditText

O Button, TextView e EditText são elementos da interface visual que são utilizados para executar algo ou apenas exibir. Todos os componentes no Android possuem métodos e atributos que possibilitam utilizar o componente com diferentes funções, como o Button, que é um elemento em que o usuário pode clicar para executar uma ação e o TextView, que exibe texto para o usuário.

O EditText é um elemento parecido com o TextView. Porém, com ele, é possível que o usuário digite ou edite um texto dentro dele. Os atributos utilizados nos componentes podem sofrer alterações de acordo com o tipo do Layout que está utilizando. No exemplo a seguir, será utilizado o LinearLayout.

Continuando o projeto Hello World (criado no capítulo anterior), serão criados esses componentes de forma a se familiarizar com sua criação.

Abra o projeto (*Open an existing Android Studio Project*, como mostra a Figura “Tela inicial Android Studio”) dentro da pasta onde o salvou; se não o encontrar, basta criar um novo com uma Activity vazia (Empty Activity). Depois, busque pelo arquivo activity_main.xml (na pasta res -> layout). No canto direito superior da nossa *Editor Window*, há três modos de visualização: Code, Split (tela dividida ao meio com as opções de Código e Design) e Design. Selecione o modo Code, como mostra a Figura “Arquivo activity main.xml”

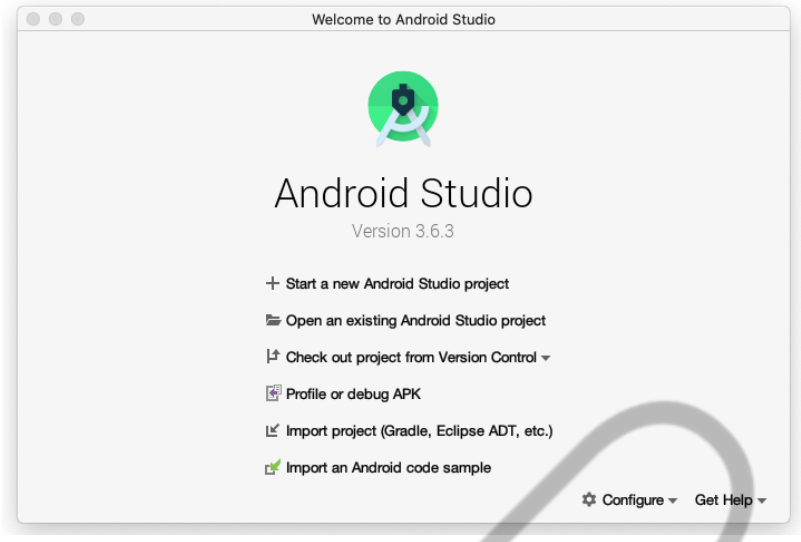

Figura 3.3 – Tela inicial Android Studio
Fonte: Elaborado pelo autor (2020)

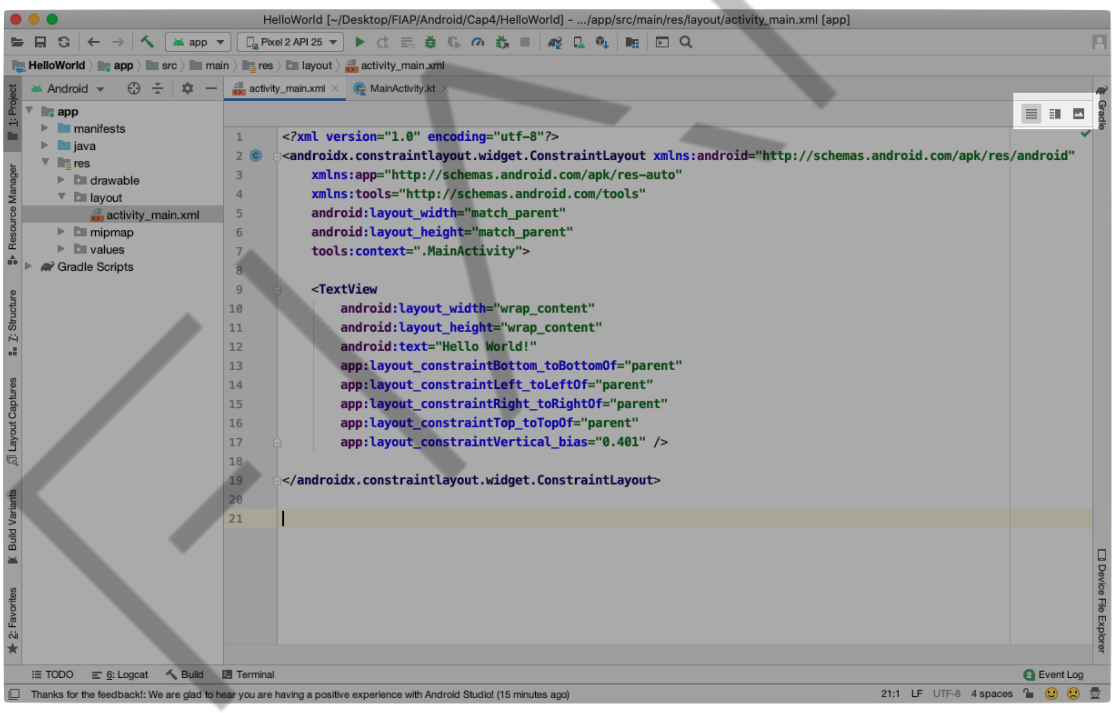

Figura 3.4 – Arquivo activity_main.xml
Fonte: Android Studio (2020)

O objetivo é deixar o layout de acordo com o exemplo do Código-fonte “Adicionando os componentes no layout e alterando o tipo”. Altere o tipo do Layout que, por padrão, vem com ConstraintLayout para LinearLayout e acrescente os componentes TextView, EditText e Button. O LinearLayout faz com que cada widget filho permita ser empilhado com orientação vertical ou horizontal. Nesse caso, o

empilhamento é vertical (android:orientation="vertical"). Também é interessante observar alguns parâmetros utilizados nos componentes para entendimento:

- layout_width e layout_height: definem as dimensões verticais e horizontais. O LinearLayout é responsivo, ou seja, ele auto se ajusta para diferentes dimensões de tela. A primeira, a match_parent, informa que o componente deve usar todo o espaço disponível na dimensão relacionada (vertical ou horizontal). Já o wrap_content tem por premissa utilizar o mínimo espaço possível;
- layout_gravity: permite informar o posicionamento relativo do componente. Nesse caso, o EditText e o TextView estão alinhados ao centro;
- textSize: define o tamanho da fonte de texto;
- margin e marginTop: definem a dimensão de margem entre componentes.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <EditText
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="Digite seu nome"
        android:textSize="24dp"
        android:layout_marginTop="60dp"
        android:layout_gravity="center"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="24dp"
        android:layout_marginTop="60dp"
        android:layout_gravity="center"/>

    <Button
        android:text="Clique aqui"
        android:textSize="20dp"
```

```
        android:layout_margin="70dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Código-fonte 3.3 – Adicionando os componentes no layout e alterando o tipo
Fonte: Elaborado pelo autor (2019)

Nas opções de visualização, no canto direito superior da Editor Window, troque a opção para *Design*. O resultado esperado é mostrado na Figura “Resultado do Código-fonte Adicionando os componentes no layout e alterando o tipo”.

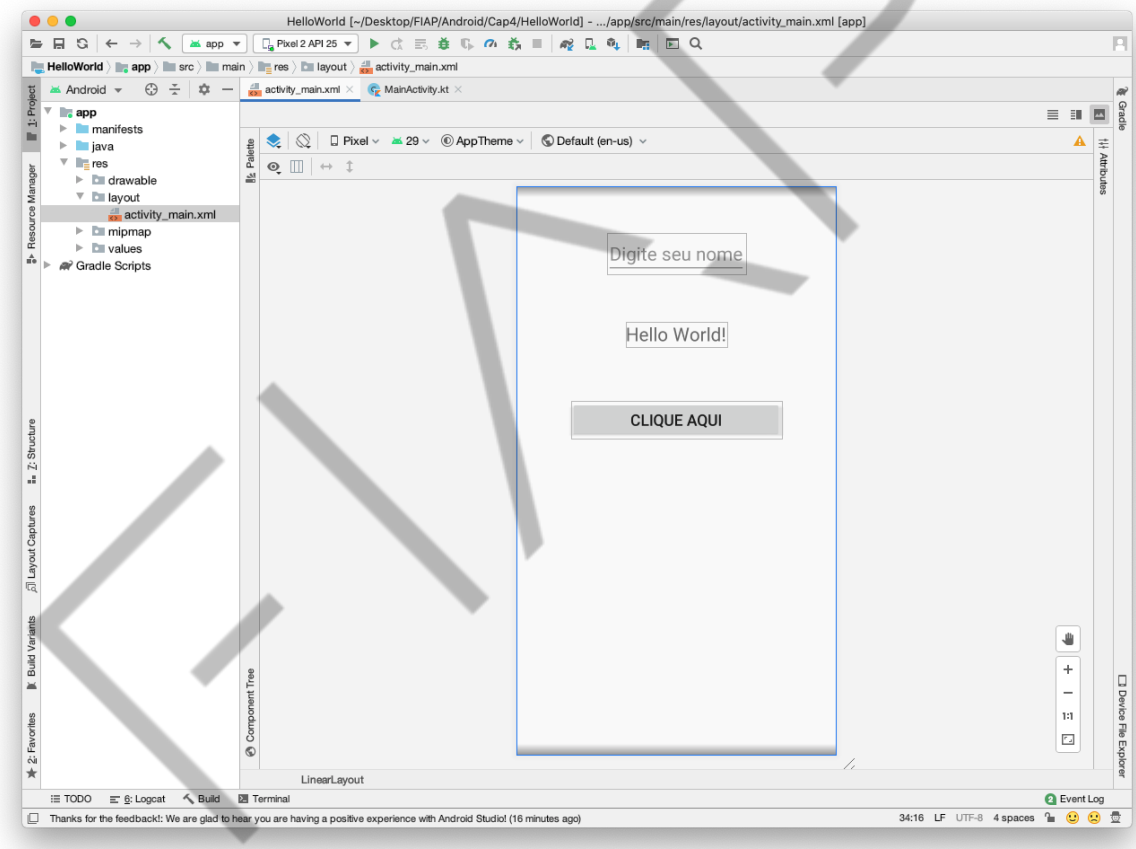

Figura 3.5 – Resultado do Código-fonte Adicionando os componentes no layout e alterando o tipo
Fonte: Android Studio (2020)

3.4 Interagindo com o usuário: OnClick (XML versus Listener)

Existem duas formas de executar ações de click no Android: diretamente no XML ou em tempo de execução com o uso do *listener*. Neste capítulo, serão apresentadas as duas formas.

Ao ser utilizado, por exemplo, o método *onClick* em uma função direta no XML, é preciso acrescentar o atributo no componente que deseja atribuir a função de clique, lembrando que vários componentes possuem esse atributo, ou seja, não podemos nos limitar em pensar apenas no componente Button, sendo que o mesmo evento pode ser vinculado em outros widgets. O Código-fonte “Atribuindo onClick a um componente no XML” mostra um exemplo do uso do método onClick em um botão.

```
...
<Button
  android:onClick="onCliqueAqui"
   android:text="Clique aqui"
   android:textSize="20dp"
   android:layout_margin="70dp"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" />
...
```

Código-fonte 3.4 – Atribuindo onClick a um componente no XML
Fonte: Elaborado pelo autor (2019)

Para executar o método onClick atribuído direto no XML, basta criar uma função com o nome definido no *onClick*. Além disso, é necessário, no código-fonte da Activity, ter um método definido com o mesmo nome adicionado na tag android:onClick (como mostra o Código-fonte “Elaborando o método onCliqueAqui executando o componente Toast”). Como exemplo de evento disparado com o clique no botão, foi utilizado o componente Toast (que exibe notificações curtas em pop-up no aplicativo Android).

```
fun onCliqueAqui(v: View?){


Toast.makeText(this,"Testando",Toast.LENGTH_SHORT).show()

     }
```

Código-fonte 3.5 – Elaborando o método onCliqueAqui executando o componente Toast
Fonte: Elaborado pelo autor (2019)

No Kotlin, a vinculação dos componentes com o código-fonte pelo método *findViewById* (como é utilizado na opção de programação em linguagem Java) não é necessária, pois a linguagem já trata isso. O id então é o nome que identifica qual componente (por meio da tag *android:id*) é em nosso XML, portanto é necessário ter atenção e colocar um id que faça referência ao componente. Por exemplo, se for um

botão, podemos colocar um prefixo *btn* que nos indica que esse componente é um botão e assim por diante. Isso depende dos *guidelines* de programação que seu projeto possa utilizar. O Código-fonte “Adicionando id aos componentes no activity_main.xml” apresenta um exemplo de inserção dos ids.

```
...
<EditText
   android:id="@+id/edt_Nome"
   ...
  />

 <TextView
   android:id="@+id/txv_Resultado"
   ...
   />

 <Button
   android:id="@+id/btn_CliqueAqui"
   ...
  />
...
```

Código-fonte 3.6 – Adicionando id aos componentes no activity_main.xml
Fonte: Elaborado pelo autor (2019)

A ação que o Button irá executar nesse exemplo é simples: recuperar o texto que o usuário digitar no EditText e apresentá-lo no TextView. Para recuperar um texto digitado pelo usuário será utilizado o método getEditableText. Esse método recupera o que foi ou será digitado dentro do EditText. Em seguida, esse valor é atribuído a uma String e posteriormente a utiliza para mudar o texto dentro do TextView. Tudo isso será executado após o usuário clicar no botão. O Código-fonte “Criando o listener e executando ações” mostra a criação do listener.

```
package com.example.android.helloworld

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        //Criando listener onClick
        btn_CliqueAqui.setOnClickListener {

            //Recuperando o texto digitado pelo usuário e
```

```
            //atribuindo a uma String
            var nome:String = edt_Nome.editableText.toString()

            //Alterando o texto dentro do TextView
            txv_Resultado.text = nome
        }
    }
}
```

Código-fonte 3.7 – Criando o listener e executando ações
Fonte: Elaborado pelo autor (2020)

Quando o usuário digitar o nome no campo solicitado e clicar no botão, o texto será substituído pelo digitado, conforme mostra a Figura “Criando o listener e executando ações”. É possível observar que os listeners, quando definidos para tal componente, têm prioridade de execução em relação ao evento de onClick vinculado à tag android:onClick. Perceba que o Toast não foi exibido no último exemplo. Para exibir o Toast, basta incluí-lo dentro do método *setOnClicIListener*.

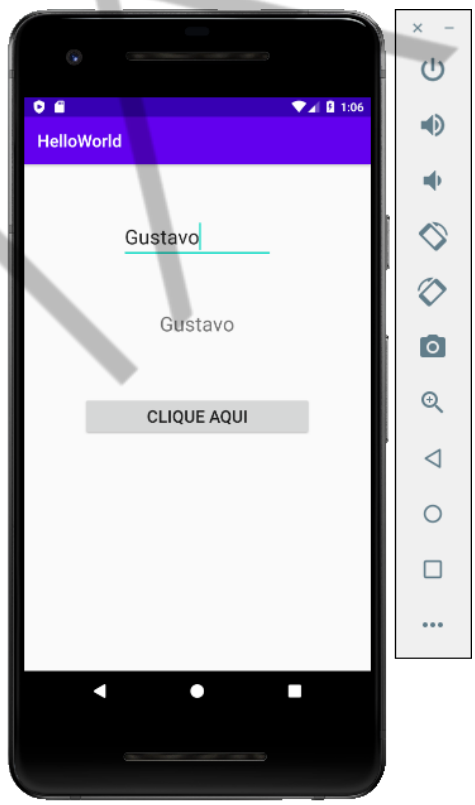

Figura 3.6 – Rodando o app no emulador
Fonte: Elaborado pelo autor (2020)

3.5 Navegação entre telas: Intent

As *Intents* são mensagens assíncronas que permitem que os componentes de um aplicativo solicitem a funcionalidade de outros componentes do Android.

Basicamente, existem dois tipos de Intents: as explícitas que definem o componente que deve ser chamado pelo sistema e as implícitas que especificam a ação que deve ser realizada e, opcionalmente, os dados que fornecem o conteúdo para a ação. Por exemplo, a partir de um componente do seu aplicativo, você pode acionar outro componente no sistema Android que gerencia fotos, mesmo que esse não faça parte do seu aplicativo. Nele, você seleciona uma foto e retorna ao aplicativo para usar a foto selecionada.

Para criar uma Intent explícita são necessários basicamente dois parâmetros: o context (Activity de origem) e o recurso de destino que pode ser uma Activity ou um Service. A única diferença entre eles é o método que vai lançar a intent: para Activity, é o método startActivity() e para services, é o método startService(). Os Códigos-fonte “Intent explicita utilizando startActivity” e “Intent explicita utilizando startService” apresentam exemplos do uso respectivamente.

```
...
val intentExplicita = Intent(this, Activity2::class.java)
startActivity(intentExplicita)
...
```

Código-fonte 3.8 – Intent explicita utilizando startActivity
Fonte: Elaborado pelo autor (2019)

```
...
val intentService = Intent(this, Service1::class.java)
startService(intentService)
...
```

Código-fonte 3.9 – Intent explicita utilizando startService
Fonte: Elaborado pelo autor (2019)

Quando uma Intent implícita é enviada para o sistema Android, ele irá procurar por todos os componentes registrados para a ação específica e o tipo de dados enviado. Caso exista apenas um componente, ele será aberto diretamente e, se forem encontrados vários componentes, o Android os abrirá em uma lista e deixará que o usuário escolha qual utilizar.

```
...
val i = Intent(Intent.ACTION_VIEW,
Uri.parse("https://www.fiap.com.br"))
// Verifica se existem aplicativos que atendem essa Intent no
Android
if (i.resolveActivity(packageManager) != null) {
startActivity(i)
}
...
```

Código-fonte 3.10 – Intent implícita passando uma URL
Fonte: Elaborado pelo autor (2019)

Um exemplo comum é a utilização das intents para compartilhar conteúdos com outros aplicativos, como redes sociais.

```
...
val sendIntent = Intent()
sendIntent.action = Intent.ACTION_SEND
sendIntent.putExtra(Intent.EXTRA_TEXT, "Texto ou URL a ser
compartilhada")
sendIntent.type = "text/plain"
startActivity(sendIntent)
...
```

Código-fonte 3.11 – Intent para compartilhar conteúdos
Fonte: Elaborado pelo autor (2019)

Agora que já sabemos o que é *Intent* e como implementá-la, vamos criar uma navegação entre duas telas no nosso aplicativo usando esse recurso.

Com o botão direito na nossa janela de projetos, selecione a opção *New*. Dentre as várias opções que aparecerão, iremos clicar na opção *Activity*. Dessa forma, o Android Studio criará tanto a classe que irá comandar a nossa tela quanto o arquivo xml para que possamos modificar o layout da nova tela; também será modificado automaticamente o nosso arquivo AndroidManifest.xml, fazendo o registro dessa nova tela. Por último, selecione a opção *Empty Activity,* como mostrado na Figura “Criando uma nova Activity”.

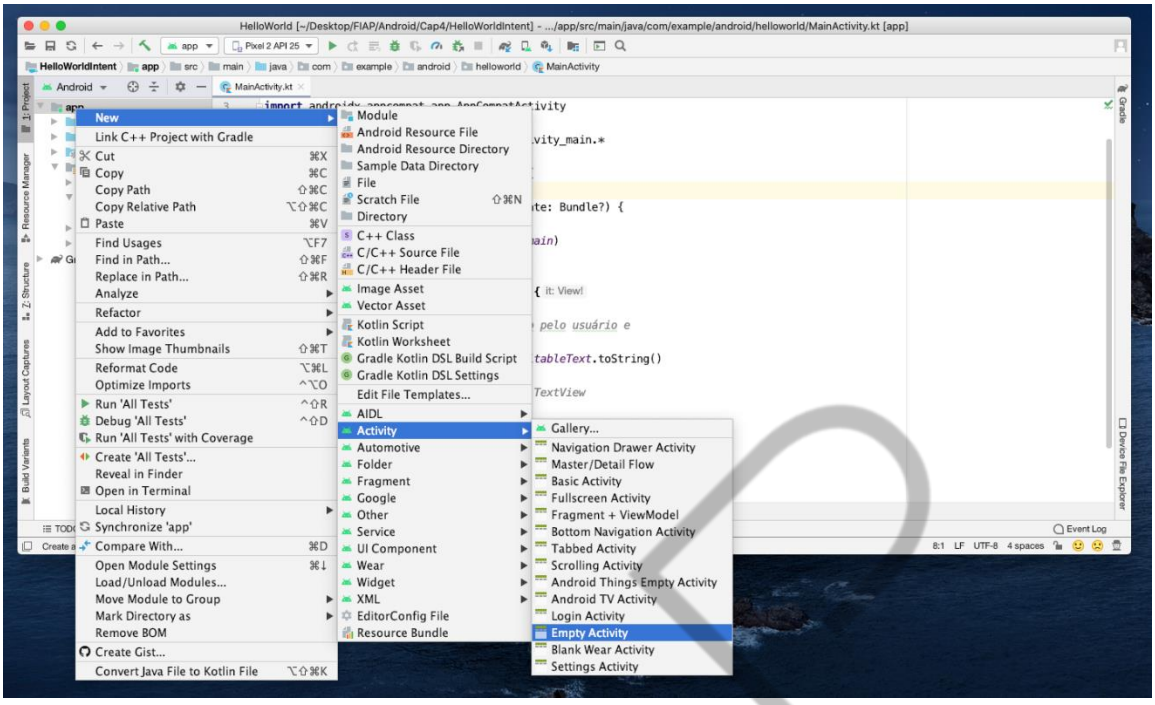

Figura 3.7 – Criando uma nova Activity
Fonte: Elaborado pelo autor (2020)

Na nova janela que aparecerá, iremos configurar nossa *Activity*. Vamos alterar o nome da nossa Activity para “ProximaActivity”, como vemos na Figura “Configurando uma nova Activity”. Repare que o campo do *layout name,* automaticamente, será alterado para “activity_proxima, que será o nome do nosso arquivo xml, criado dentro da pasta *res/layout*.

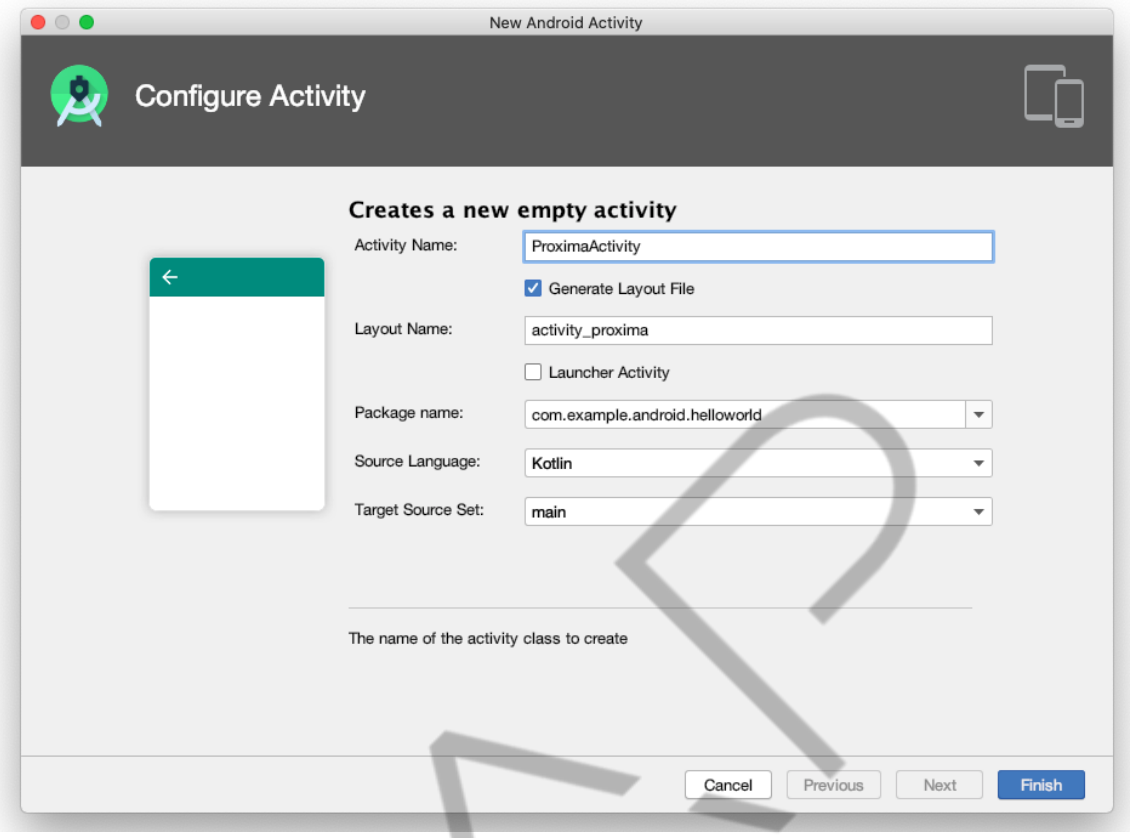

Figura 3.8 – Configurando uma nova Activity
Fonte: Elaborado pelo autor (2020)

Feito isso, vamos modificar a classe MainActivity.kt. Nessa classe, temos o listener do btn_CliqueAqui que, sempre que tiver o evento do clique no botão, será chamado. Dentro dele, vamos criar uma *Intent* passando como parâmetro o nosso Context e a Classe da segunda Tela, como apresentado no Código-fonte “Intent para próxima tela”. Dessa forma, sempre que o usuário clicar no botão “Clique Aqui”, nossa Activity será apresentada.

```
btn_CliqueAqui.setOnClickListener {

    //Recuperando o texto digitado pelo usuário e
    //atribuindo a uma String
    val nome:String = edt_Nome.editableText.toString()

    //Alterando o texto dentro do TextView
    txv_Resultado.text = nome

    val intent = Intent(this, ProximaActivity::class.java)
    startActivity(intent)

}
```

Código-fonte 3.12 – “Intent para próxima tela”
Fonte: Elaborado pelo autor (2020)

Parabéns! Agora você já conhece os primeiros passos para criar um aplicativo Android com múltiplas telas.

REFERÊNCIAS

ANDROID. **Guia do Usuário**. 2020. Disponível em: <https://developer.android.com/?hl=pt-br>. Acesso em: 06 out. 2020.

ANDROID. **Reference Documentation** 2020. Disponível em: <https://developer.android.com/reference/kotlin/android/app/Activity>. Acesso em: 06 out. 2020.

ANDROID. **Reference Documentation** 2020. Disponível em: <https://developer.android.com/training/basics/firstapp/starting-activity> Acesso em: 06 out. 2020.

LECHETA, R. R. **Android Essencial com Kotlin**. São Paulo: Novatec, 2017.